Curso práctico del Linux - Parte II

por: Claus Denk

Marzo 1997

Universidad de Sevilla - Facultad de Física

Índice General

.

1 Introducción

En diciembre de 1995 se organizó un curso sobre el Linux en la Facultad de Física de la Universidad de Sevilla. La intención de ese curso era proporcionar una ayuda para instalar y configurar este sistema operativo. En esta segunda parte del curso voy a intentar explicar algunos conceptos que nos pueden ser de gran utilidad a la hora de usar el Linux. Para poder aprovechar estas clases al máximo debemos

- Tener a nuestro alcance un ordenador con el Linux instalado.

- Entender los comandos básicos del Unix.

Los temas generales de este curso serán:

- Configuración e instalación de aplicaciones. Adaptación de algunas aplicaciones al español. Cómo instalar programas que no están incluidos en nuestra distribución de Linux. Cómo compilar un programa que sólo está disponible en código fuente.

- Aplicaciones. Hay muchas aplicaciones que pueden ser muy útiles, pero muchas veces ni siquiera sabemos que existen (`make`, `gawk`, `perl`, ...).

- Redes. Un servidor de la WWW nos permite publicar en la Internet. Cómo montar un servidor de ficheros. Computación paralela en un "cluster" de estaciones de trabajo Linux.

Los ejemplos se darán al estilo "receta", lo cual significa que sólo serán aplicables (estrictamente) cuando se use la distribución Slackware 3.1.

Notas:

¿ Qué hay de nuevo en el mundo del Linux ?

En el último año, el esfuerzo de los programadores ha hecho posible:

- Conseguir un Linux/x86 muy estable.

- Muchas aplicaciones disponibles para Linux.

- Linux para Alpha, SPARK, 68000 y PowerMacintosh (MkLinux).

- Aplicaciones comerciales.

Organización de estas notas

Al principio me hubiera gustado continuar las notas de la primera parte del curso (empezando por la página 66), pero he decidido separar estas notas de las anteriores ya que existen diferencias entre las distribuciones del Slackware 3.0 (la que se usó en el primer curso) y 3.1/December 96 (la que usaremos en este).

Notas:

2 Adaptación de algunas aplicaciones al Español

La existencia de caracteres especiales (ñ, tildes, diéresis) en muchos idiomas puede presentar un problema a la hora de usar un sistema operativo o una aplicación. Por defecto, la mayoría de los programas están configurados para los países de habla inglesa y no podemos usar caracteres especiales con ellos.

El problema proviene de los "primeros días" de la informática. Para codificar un carácter, se inventó la tabla ASCII (1963, American Standard Code for Information Interchange) con sus 128 caracteres, esta tabla aún sigue siendo estándar para muchos programas (podemos consultar la tabla ASCII con el comando `man 7 ascii`). Después han aparecido varias extensiones de este estándar, siendo la más importante la ISO 8859-1, también llamada Latin1 o IsoLatin (véase `man 7 iso_8859_1`). Estas extensiones son de "8-bits", ya que en ellas solo se añaden al ASCII los códigos 128-255 (decimal).

Para que un programa pueda trabajar con los caracteres especiales, debe usar una tabla de codificación de 8-bits que contenga todos los caracteres necesarios para nuestro idioma.

Notas:

Uso del teclado español con el Linux

Durante la instalación del Linux ya hemos realizado dos configuraciones:

- Hemos elegido el "keymap" `es.map` (véase fichero `/etc/rc.d/rc.keymap`)
- Hemos configurado el X-Windows para un teclado español (fichero `XF86Config`)

La versión 3.2 del X-Windows permite usar las teclas mudas con las siguientes definiciones en el `XF86Config` (sección "Keyboard", variable "XkbDisable" no definida):

```
Xkbkeycodes "xfree86"
XkbTypes    "default"
XkbCompat   "default"
XkbSymbols  "en_US(pc101)+es"
XkbGeometry "pc"
```

Con la versión 3.12, para poder usar las tildes en el X-Windows, hacia falta utilizar el comando `xmodmap fichero_de_entrada` con un fichero de entrada algo parecido a

```
keycode  34 = grave asciicircum bracketleft
keycode  38 = a A aacute Aacute
keycode  26 = e E eacute Eacute
keycode  31 = i I iacute Iacute
keycode  32 = o O oacute Oacute
keycode  30 = u U uacute Uacute
keycode  48 = apostrophe Ddiaeresis braceleft
```

Para crear este fichero son útiles:

- El Programa `xev`, para averiguar los "keycodes".
- El fichero `/usr/X11/include/X11/keysymdef.h`, para averiguar los nombres de los caracteres.

Notas:

Emacs

Para usar el Emacs (Editor MACroS) con caracteres de la tabla "latin1", hay que crear un fichero de configuración para el Emacs. Este fichero se llama `.emacs` y reside en el `$HOME` del usuario. Creamos un fichero con este nombre, añadimos la línea

```
(standard-display-european t)
```

y guardamos este fichero. Al arrancar el Emacs de nuevo, podremos usar los caracteres especiales.

Telnet y rlogin

Si nos conectamos con otra máquina, los caracteres especiales tienen que "pasar" por la red. El comando `telnet` no deja pasar estos caracteres, mientras que el `rlogin` sí lo permite.

Sugerencia:

Podemos crear un fichero de configuración, llamado `.rhosts` para facilitar el acceso a otras máquinas que usemos frecuentemente. Creamos este fichero en la cuenta de la máquina a la cual queremos acceder y ponemos líneas de la forma

```
máquina1_de_donde_accedemos usuario
máquina2_de_donde_accedemos usuario
```

en este fichero. Por ejemplo, el siguiente fichero `.rhosts`, que reside en el `$HOME` del usuario `denk` de la máquina `obelix.cica.es`

```
numerix.us.es denk
servix.us.es  denk
servix.us.es  alos
```

permite el acceso a la cuenta del `denk` de `obelix.cica.es` al usuario `denk` de las máquinas `numerix.us.es` y `servix.us.es`. Además, el usuario `alos` de `servix.us.es` puede acceder a esta cuenta sin necesidad de teclear el `password` cuando se realiza un `rlogin` al `obelix.cica.es`. Por razones de seguridad, lo último no es muy aconsejable.

Notas:

Latex

Para usar el Latex directamente con los caracteres especiales, sin tener que usar secuencias crípticas como

```
El pa\´{\i}s
```

podemos utilizar el estilo "t1enc" del Latex. El estilo "spanish" se encarga de poner "Capítulo" en vez de "Chapter". En el modo de compatibilidad con el Latex 2.0.9 un fichero tendría la siguiente forma:

```
\documentstyle[spanish,t1enc,12pt]{report}
\begin{document}
\chapter{test}
Hola, el acentó ésta mal puesto. La ñ es una pequeña dificultad.
La homeopatía de la separación en español !
\textquestiondown una formula ? $x ^{\prime}= y´$
\end{document}
```

Un fichero para el Latex2e tendría la siguiente forma:

```
\documentclass[12pt]{report}
\usepackage[spanish]{babel}
\usepackage{isolatin1}
\begin{document}
\chapter{test}
Hola, el acentó ésta mal puesto. Y la ñ es una pequeña dificultad.
La homeopatía de la separación en español !
¿ una formula ? $x ^{\prime}= y´$
\end{document}
```

Notas:

Si compilamos el primer documento por primera vez en nuestra máquina, podemos encontrarnos con un extraño mensaje de error:

```
This is METAFONT, Version 2.718 (C version 6.1)

kpathsea: Running MakeTeXMF dcr1200.mf
/bin/mkdir: cannot make directory `source´: Permission denied
/usr/lib/teTeX/bin/MakeTeXmkdir: could not create directory /var/texfonts/source
```

Este error ha aparecido con la distribución 3.1 del Slackware que estamos usando aquí,y podemos arreglarlo de manera fácil (como `root`):

```
cd /var/texfonts
mkdir source
chmod a+xrwt source
```

El programa que se encarga de crear las fuentes para el Latex necesita el directorio `/var/texfonts/source` para crear ciertos tipos de fuentes. El parámetro "t" del comando `chmod` activa el "sticky bit" de este directorio, esto significa que un usuario no privilegiado no puede borrar o cambiar el nombre de un fichero de otro usuario.

Notas:

Ahora que el Latex funciona, nos damos cuenta de que la separación de las palabras en español no funciona correctamente. Si miramos la salida del Latex, veremos:

```
This is TeX, Version 3.14159 (C version 6.1)
(span.tex
LaTeX2e <1996/06/01>
Hyphenation patterns for english, german, loaded.
(/usr/lib/teTeX/texmf/tex/latex/base/latex209.def
Entering LaTeX 2.09 compatibility mode.
..
```

El programa ha cargado una tabla de separación de "english" y otra de "german", pero no aparece el "spanish". Para conseguir esto, tenemos que configurar un poco el Latex. La manera de proceder depende de la distribución del Latex que tengamos. En el Slackware 3.1 está incluida la distribución "teTeX" (Thomas Esser TeX), así que vamos a usar esta distribución en el ejemplo:

- Editamos el fichero `/usr/lib/teTeX/texmf/tex/generic/config/language.dat`. En este fichero quitamos de delante de la siguiente línea el %

  ```
  %spanish sphyph.tex
  ```

- Ahora hay que correr el programa `initex`. Podemos realizar esto como un usuario normal en nuestra cuenta (por ejemplo, en el directorio `/home/denk/mytex`):

  ```
  initex /usr/lib/teTeX/texmf/tex/latex/base/latex.ltx
  ```

  Este paso crea un fichero `latex.fmt` que contiene las definiciones necesarias para el comando `latex`. Este comando realmente es un "link" simbólico al comando `virtex`, que carga el fichero de formato adecuado (en el caso del Latex el `latex.fmt`, en el caso del Amstex el `amstex.fmt`).

- Hacemos una copia de seguridad del fichero original y lo reemplazamos por el `latex.fmt` que hemos creado en el paso anterior:

  ```
  cd /usr/lib/teTeX/texmf/web2c
  cp latex.fmt latex.fmt.old
  cp /home/denk/mytex/latex.fmt .
  ```

Ahora probamos el Latex "nuevo":

```
idefix3:~$ latex span.tex
This is TeX, Version 3.14159 (C version 6.1)
(span.tex
LaTeX2e <1996/06/01>
Hyphenation patterns for english, german, spanish, loaded.
...
```

Ya nos debe funcionar correctamente la separación en español. Hemos realizado esta configuración "a mano". La distribución del teTeX incluye un programa que permite configurar el Latex cómodamente (`texconfig`).

Notas:

3 Instalación de una distribución binaria

En esta sección vamos a explicar cómo se instala un programa o una librería que hayamos obtenido en forma binaria. Puesto que la gran mayoría de los programas están disponibles en esta forma, esto nos facilitará la instalación.

¿ Cuándo hace falta instalar algo ?

- Hemos borrado equivocadamente algunos ficheros importantes.
- Queremos un programa nuevo.
- Queremos una versión más actual de un programa que ya tenemos.

Sitios en la red donde se pueden encontrar distribuciones binarias

- El sitio oficial:
  `ftp://sunsite.unc.edu/pub/Linux`
- Algún "mirror" de este sitio, por ejemplo,
  `http://www.cs.us.es/archive/linux.html`
- Páginas de la WWW con "links", como por ejemplo
  `http://www-ocean.tamu.edu/~baum/linuxlist.html`
- Varios sitios de `ftp` localizables con el `xarchie`

Notas:

Una pequeña selección de programas útiles

- Acceso a la red: `netscape` y `xarchie`.
- Análisis gráfico de datos: `plotmtv` y `xmgr`.
- Animación: `xanim`.
- Cálculo numérico: `octave`
- Utilidades: `recode`.
- Fortran: `ftnchek` y código del Netlib (`http://www.netlib.org/`).

Tipos de ficheros

El primer paso siempre consiste en transferir a la máquina local los ficheros necesarios, usando para ello el comando `ftp`. Una distribución binaria normalmente consiste en dos ficheros:

- `fichero.lsm` - Descripción del programa.

- `fichero.tar.gz` o `fichero.tgz` - archivo comprimido que contiene los ficheros.

Una vez obtenidos estos ficheros, podemos empezar con la instalación. En algunos casos no encontraremos un fichero `.lsm` que acompañe al paquete.

Notas:

Ejemplo de una instalación

Aquí realizaremos una instalación sencilla paso a paso. ¡ Ojo, los pasos que hay que dar dependen del paquete que queramos instalar! ¡Lo siguiente no se puede utilizar como una receta general válida para todos los programas !

En este ejemplo vamos a instalar el programa "xarchie". Podremos usar este progama más adelante para buscar otros programas que nos interesen. Tenemos que decidir dónde (en qué directorio) vamos a instalar el programa. Ya que se trata de un programa que no es parte del sistema operativo, debemos separarlo de los ficheros del sistema operativo. El directorio `/usr/local` está pensado para esto, y, si estamos muy seguros, podemos instalar el programa directamente en este directorio como usuario `root`. Aun estando seguro, es aconsejable instalar el programa primeramente en un directorio de usuario, ya que así evitaremos errores fatales que pueden cometerse como usuario `root`(☹). Es conveniente, por tanto, ir por el camino seguro realizando los siguientes pasos como un usuario "normal".

Hemos obtenido los ficheros `xarchie2.0.9.lsm` y `xarchie2.0.9+source.tgz`. Creamos un directorio y copiamos los dos ficheros a este directorio:

```
mkdir myinst
cp  xarchie2.0.9* myinst
cd myinst
ls
```

Ahora leeremos detenidamente el fichero `.lsm`:

```
vi xarchie2.0.9.lsm
```

Podemos proceder a descomprimir (y, en este caso, a extraer todos los ficheros del archivo):

```
tar -zxvf xarchie2.0.9+source.tgz
```

Este comando creará un directorio `xarchie-2.0.9`. Cambiaremos el directorio de trabajo a este directorio y leeremos todos los ficheros que tienen "pinta" de documentación:

```
cd xarchie-2.0.9
ls
vi README
vi INSTALL
```

En este caso concreto encontraremos también un ejecutable `xarchie` junto al código fuente. No hace falta compilar este programa y podemos probarlo directamente:

```
./xarchie&
```

Esto no funciona con todos los programas, ya que algunos requieren bibliotecas instaladas en "su" sitio. Para averiguar como debemos instalar el `xarchie`, leeremos el fichero `INSTALL`:

```
6. Install the package using
        % make install install.man
   This will install the xarchie executable, the app-defaults file,
   and the manpage.

NOTE: The Imakefile provides the target "World", which does the following:
        % make -n World
        make Makefile
        make Makefiles
        make depend
        make clean
        make
```

Este paquete está preparado para compilarlo primero y depués instalarlo. El comando

```
make install install.man
```

empieza por compilar los fuentes del programa y queremos precindir de este paso.

Notas:

Para instalar el ejecutable directamente, abriremos el fichero `Makefile` y buscaremos la palabra "install". De esta manera, encontraremos la siguiente información:

```
...

INSTALL = install

...

XAPPLOADDIR = $(LIBDIR)/app-defaults

...

# Where do you want this stuff? Uncomment and adjust these to change the
# destinations of "make install" and "make install.man" if the defaults
# are not satisfactory.
#BINDIR = bin
#MANDIR = man/man1
# Use this for R5
#MANSUFFIX = 1a

...

install:: xarchie
      @if [ -d $(DESTDIR)$(BINDIR) ]; then set +x; \
      else (set -x; $(MKDIRHIER) $(DESTDIR)$(BINDIR)); fi
      $(INSTALL) -c $(INSTPGMFLAGS)  xarchie $(DESTDIR)$(BINDIR)

install.man:: xarchie.man
      @if [ -d $(DESTDIR)$(MANDIR) ]; then set +x; \
      else (set -x; $(MKDIRHIER) $(DESTDIR)$(MANDIR)); fi
      $(INSTALL) -c $(INSTMANFLAGS) xarchie.man $(DESTDIR)$(MANDIR)/xarchie.
      $(MANSUFFIX)
install:: Xarchie.ad
      @if [ -d $(DESTDIR)$(XAPPLOADDIR) ]; then set +x; \
      else (set -x; $(MKDIRHIER) $(DESTDIR)$(XAPPLOADDIR)); fi
      $(INSTALL) -c $(INSTAPPFLAGS) Xarchie.ad $(DESTDIR)$(XAPPLOADDIR)/Xarchie
```

Notas:

Cuando tecleamos `make install` , el programa `make` lee la información necesaria del fichero `Makefile` y "ejecuta" las reglas para `install`. La primera regla en el ejemplo es:

- Compila el programa `xarchie`
- Si no existe el directorio `$(DESTDIR)$(BINDIR)`, crea este directorio
- Usa el programa `$(INSTALL)` para instalar el fichero `xarchie`

La variable `$(INSTALL)` ha sido definida como `install`, podemos ver mediante el comando `man install`, que es lo que hace este comando.

De esta manera podemos averiguar los pasos necesarios para instalar un programa, en nuestro caso concreto podemos proceder de la manera siguiente:

```
# cambiar al super-usuario
su
# a ver que hay ya en el directorio de destino
ls /usr/local/bin
# copiar el ejecutable
cp xarchie /usr/local/bin
# a ver que hay en el directorio del man
ls ls /usr/local/man/man1
# copiar la pagina del manual
cp xarchie.man /usr/local/man/man1/xarchie.1
# donde estan los app-defaults ?
find / -name app-defaults
# a ver que ficheros hay
ls /var/X11R6/lib/app-defaults
# copiar los defaults
cp Xarchie.Ad /var/X11R6/lib/app-defaults/Xarchie
# volver a la cuenta del usuario
exit
# y probarlo
cd
xarchie&
man xarchie
```

Consideraciones:

- Hay que probar el programa detenidamente para asegurar que funciona bien

- Antes de instalar un programa en el `/usr/local`, podemos dejarlo instalado en el directorio de un usuario y crear un `link` en el `/usr/local/bin` a este programa. Podemos dejarlo allí durante una fase de pruebas.

- Hay que decirle a los usuarios que hay un programa nuevo disponible

- Conviene apuntar que programas hemos instalado en un sistema

- De vez en cuando podemos mirar en el sitio donde hemos obtenido el programa, a ver si ya hay una versión mas nueva. ¡ La gran mayoría de los programas para Linux están evolucionando continuamente !

- Si estamos contento con el programa, se lo diremos a otras personas para que puedan instalarselo también.

Notas:

4 La utilidad make y la compilación de programas

El fin de esta sección es entender los pasos fundamentales que hay que dar para compilar un programa. La utilidad `make` es una herramienta imprescindible para esta tarea. Además, podemos utilizar el `make` para nuestros propios proyectos de programación.

Guia rápida para el make

La utilidad `make` "organiza" la compilación de programas consistentes en varios ficheros fuentes. Para determinar, cómo hay que compilar un programa, escribiremos un fichero "Makefile". El `make` lee este fichero y arranca los compiladores necesarios para compilar las fuentes.

Sólo vamos a explicar este tema brevemente. Para obtener más información acerca del `make` se puede usar el comando `info make`. El primer paso para poder usar el `make` es crear un fichero "Makefile". Vamos a explicar los "ingredientes" de este fichero mediante un pequeño programa que hemos escrito en C y que consiste en el siguiente código:

```
File main.c:
------------
#include "mydefs.h"
#include "myfunc.h"
#include "myprint.h"

int main(){
   static double V[NDIM],val;
   int i;
   for (i=0;i<NDIM;i++) {
     val = i*0.1;
     V[i] = ener(val);
   }
   myprint(V);
   return 0;
}
```

```
File mydefs.h:
--------------
#define NDIM 100
#define FACTOR 1.2345

File myfunc.c:
--------------
#include <stdio.h>
#include "mydefs.h"
#include "myfunc.h"

double ener(double a) {
  return FACTOR*a*a;
}

File myfunc.h:
--------------
extern double ener(double a);

File myprint.c:
---------------
#include <stdio.h>
#include "mydefs.h"
#include "myprint.h"

void myprint(double a[]) {
  int i;
  for (i=0;i<NDIM;i++)
    printf("index: %d value: %e\n",i,a[i]);
}

File myprint.h:
---------------
extern void myprint(double a[]);
```

Para compilar el programa podemos utilizar el comando

```
gcc main.c myfunc.c myprint.c
```

Este procedimiento se hace poco practico en cuanto el tamaño del programa aumenta, ya que tenemos que compilar todas las fuentes cada vez que queremos

crear el ejecutable. Podemos usar los ficheros "Makefile" para una compilación "inteligente" (compilar solamente los ficheros necesarios) ya que un cambio en un fichero no requiere necesariamente la compilación de todas las fuentes.

Los ficheros "Makefile" contienen:

- Reglas (rules). Se especifica un "target", una regla que determina cuando el target requiere actualización y uno o varios comandos para realizar esta actualización. Ejemplo:

```
myfunc.o: myfunc.c myfunc.h mydefs.h
        gcc -c myfunc.c
```

  En el ejemplo, el target es `myfunc.o`. El target require actualización si no existe o si la fecha del fichero es anterior a la fecha de los ficheros `myfunc.c`, `myfunc.h` y `mydefs.h`. La actualización consiste en la compilación del fichero `myfunc.c` mediante el compilador `gcc`.

- Declaraciones de variables. Podemos declarar variables y usarlas en el "Makefile". Ejemplo:

```
CFLAGS= -Wall -pedantic -O2
OBJ = main.o myfunc.o myprint.o

myprog: $(OBJ)
        $(CC) -o myprog $(OBJ)

clean:
        rm -f myprog $(OBJ)
```

  En este ejemplo declaramos la variable `OBJ` como una lista de ficheros. El target `myprog` depende de esta lista, y su compilación también usa esta lista. El target `clean` no depende de otro fichero, la acción para este target es el comando `rm` que borra el programa `myprog` y todos los ficheros de la lista `OBJ`. En este ejemplo no hemos definido ninguna regla para los miembros de la lista `OBJ`. En este caso el programa `make` utiliza "reglas por defecto": Si `make` encuentra los ficheros `main.c`, `myfunc.c` y `myprint.c` en el directorio actual, `make` procederá a compilar estas fuentes automáticamente. El compilador de C usará los "flags" de la variable `CFLAGS`.

- Comentarios. Una línea que empieza por "#" se considera como comentario.

Notas:

Además de estos "ingredientes", los ficheros "Makefile" pueden contener condicionales, incluir otros ficheros, etc.

Podemos usar los ficheros "Makefile" para nuestros programas, sean escritos en C, FORTRAN, Pascal, u otro lenguaje de programación. Además de compilar nuestros programas tambien se pueden usar estos ficheros para simplificar procedimientos frecuentes, como, por ejemplo:

```
CFLAGS= -Wall -pedantic -O2
OBJ= main.o myfunc.o myprint.o

myprog: $(OBJ) mydefs.h
        $(CC) -o myprog $(OBJ)

myfunc.o: myfunc.c myfunc.h mydefs.h
        $(CC) $(CFLAGS) -c myfunc.c

myprint.o: myprint.c myprint.h mydefs.h
        $(CC) $(CFLAGS) -c myprint.c

main.o: main.c mydefs.h myfunc.h myprint.h
        $(CC) $(CFLAGS) -c main.c

clean:
        rm -f myprog $(OBJ)

demo:   myprog
        rm -f res.dat plot.in
        myprog > res.dat
        echo ´plot "res.dat" using 4 with lines´ > plot.in
        echo ´pause -1 "Press Return"´ >> plot.in
        gnuplot plot.in
```

Para averiguar las "dependecias" de cada fichero, podemos usar el compilador `gcc` con el switch "-M" o "-MM":

```
gcc -MM myprint.c
myprint.o: myprint.c mydefs.h myprint.h
```

La utilidad `gccmakedep` añade automáticamente estas dependencias al fichero "Makefile" del directorio actual.

Notas:

Ficheros Makefile de programas complejos

Un "Makefile" de un programa complejo suele ser bastante largo y no es necesario entender todo lo que se define en el fichero. Hay varios tipos de variables que se definen a menudo en los ficheros "Makefile". Aquí citamos algunos ejemplos:

- Nombres de directorios, por ejemplo:

```
MANDIR = man/man1
```

- Nombres de bibliotecas, por ejemplo:

```
XSSLIB = -lXss
```

  Estas bibliotecas se usarán a la hora de "linkar" el código

- Definiciones que se pasan al programa a la hora de compilar, por ejemplo:

```
PDEBUG = -DDEBUG
```

- Especificaciones acerca de dónde residen ficheros, por ejemplo:

```
DIR_INC = -I$(DIR_DIR)
DIR_LIB = -L$(DIR_DIR) -lDir
```

- Opciones para el compilador, por ejemplo:

```
CXXDEBUGFLAGS = -O2 -fno-strength-reduce -m486
```

Si una compilación falla, inspeccionaremos el mensaje de error e intentaremos averiguar la razón en el "Imakefile" o en el "Makefile". Muchas veces simplemente falta una biblioteca, o bien el compilador no encuentra un fichero "include" (ficheros `*.h`).

Notas:

Compilación con ficheros Makefile - ejemplo xarchie

Si el programa deseado no está disponible en forma binaria para nuestro sistema, tenemos que compilar las fuentes. La compilación de la mayoría de los programas está automatizada, y, siguiendo las instrucciones, normalmente no surgen problemas. Los pasos en una compilación son:

- Configuración. Los ficheros necesarios para la compilación son configurados para el sistema operativo en cuestión. Muchas veces con el programa esta incluido un shell script `configure`. Si ejecutamos este script con `/.configure`, la configuración se realizará de forma automática. Algunos programas requieren una configuración manual, en este caso seguiremos las instrucciones de instalación.

- Preparación de los ficheros "Makefile". Usando la información obtenida en el paso anterior, se crean los ficheros que determinan como se ha de compilar el programa. Para esta tarea pueden utilizarse las utilidades `xmkmf` y `imake`. Los ficheros "Imakefile" contienen la información necesaria para crear los ficheros "Makefile".

- Compilación con la utilidad `make`. Esta utilidad lee los ficheros "Makefile" y ejecuta todos los pasos necesarios para la compilación.

Si el programa solamente viene con un fichero "Makefile", no hay que realizar los dos primeros pasos.

Notas:

En la sección anterior hemos instalado el programa "xarchie" sin compilar las fuentes, ya que la distribución incluía el fichero ejecutable. Ahora vamos a compilar las fuentes para producir este ejecutable. Seguiremos los pasos que se indican en el fichero `INSTALL`:

```
# nos cambiamos al directorio del xarchie
cd xarchie-2.0.9
# y borramos el ejecutable que venia con la distribución
rm xarchie
# configuramos ...
sh ./configure
# creamos el Makefile
xmkmf
# y los Makefiles en los subdirectorios
make Makefiles
# actualizamos las dependencias
make depend
# compilamos las fuentes
make
# y, si queremos, lo instalamos
su
make install install.man
exit
```

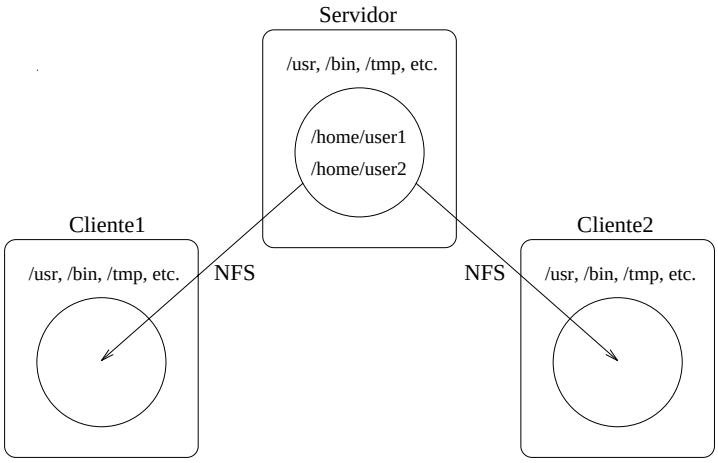
Durante la compilación pueden aparecer mensajes de aviso ("warning"). Estos mensajes no indican necesariamente un problema, pero en caso de que el programa no funcione pueden dar una pista para encontrar el error.

Notas:

5 Un Servidor de ficheros

En un grupo de trabajo muchas veces es ventajoso tener los ficheros de todos los usuarios en una estación de trabajo central. Esta estación de trabajo se llama "servidor" (de ficheros). Las estaciones de trabajo que solicitan un servicio al servidor se llaman "clientes".

Ejemplo



En este ejemplo los clientes tienen sus propios directorios para los ficheros del sistema (programas, mantenimiento del sistema, etc.), pero el directorio donde residen los ficheros de los usuarios está físicamente localizado en el servidor. Sin embargo, estos directorios se ven como si se encontraran en la máquina local. A la hora de acceder a un fichero, el cliente manda un pedido al servidor, el servidor accede al disco y devuelve el fichero al cliente (por la red). Este proceso es transparente para el usuario y es controlado por el NFS (=Network File System).

Notas:

Ventajas de usar un servidor de ficheros

- Mantenimiento mínimo. (copias de seguridad, instalación central de aplicaciones, etc.)

- Independencia de las estaciones de trabajo. Cada usuario puede trabajar en todas las estaciones de trabajo, al hacer un login siempre se "ve" el mismo directorio.

- Posibilidad de trabajar en la máquina menos cargada vía `telnet`, también se puede trabajar en varias máquinas a la vez.

- Posibilidad de dedicar una máquina adecuada a servir ficheros (disco duro grande, sistema de alimentación de emergencia), así no es necesario tener discos duros grandes en todas las maquinas.

Consideraciones

- El acceso por la red es más lento que el acceso local.

- Sobrecarga de un servidor (muchos accesos a la vez).

- Los ficheros no están accesible si no hay conexión con el servidor (problemas de red, servidor colgado, etc.)

Realización

Preparación:

- Abrir las cuentas para los usuarios en el servidor (comando `adduser`)

- Pedirle a todos los usuarios que vayan trayendo todos sus ficheros por `ftp` al servidor. Comentar, que a partir de ese momento los ficheros locales no estarán accesibles (pequeña mentira). No dejar entrar a los usuarios (fichero `/etc/nologin`) en las estaciones de trabajo mientras se realiza lo siguiente.

- Mirar los UID (user identification) en el fichero `/etc/passwd` del servidor. Los UID de los usuarios para todos los clientes deben ser identicos a los UID del servidor. Para lograr esto hay que cambiar el "owner" (dueño) de todos los ficheros y directorios de cada usuario en todos los clientes (comando `chown`). El superusuario se situará en el directorio "home" (por ejemplo `cd /home`) y cambiará recursivamente el owner de todos los ficheros (por ejemplo, `chown -R 601:100 user1`, verificar con `ls -l`). Además existen ficheros de los usuarios en el directorio `/var/spool/mail`. También hay que cambiar el owner de estos ficheros. El comando `find` puede ser usado para encontrar todos los ficheros de un usuario, usándolo desde el directorio principal en la forma `find / -uid nnn`. ¿ Cómo podríamos combinar los comandos `find` y `chown` para cambiar el owner de todos los ficheros de un usuario con una línea de comando ?

```
find / -uid nnn -exec chown mmm:kkk {} \;
```

- Ahora hay que cambiar los UID de los usuarios en los clientes, editando el fichero `/etc/passwd`. Antes de seguir podemos ofrecerles a los usuarios la posibilidad de que vean sus ficheros locales una última vez (otra pequeña mentira)

Preparar el servidor

- Editamos el fichero `/etc/exports` en el servidor y añadimos una línea para todos los clientes, por ejemplo:

```
/home cliente1.us.es(rw)
/home cliente2.us.es(rw)
```

- Ahora hay que asegurar que el servidor arranque los demonios que se encargarán de servir los ficheros a los clientes. Hacen falta los demonios `rpc.portmap`, `rpc.mountd` y `rpc.nfsd`. En el fichero `/etc/rc.d/rc.inet2` hay entradas de la forma

```
# if [ -f ${NET}/rpc.mountd ]
#       then
#       echo -n " mountd"
#       ${NET}/rpc.mountd
# fi
```

  Hay que quitar los comentarios para que quede

```
if [ -f ${NET}/rpc.mountd ]
     then
     echo -n " mountd"
     ${NET}/rpc.mount
fi
```

- Una vez realizado esto, rebotamos el servidor. En los mensajes de arranque saldrán los demonios que hemos añadido y el servidor estará preparado para recibir los pedidos de los clientes.

Preparar los clientes

Los clientes tienen que "montar" el directorio `/home` del servidor en su árbol de directorios. Un directorio de otro ordenador normalmente se monta en un directorio vacío del cliente. Montaremos el `/home` del servidor encima de los `/home` de los clientes, escondiendo así los ficheros ya existentes en los clientes. A largo plazo, debemos eliminar estos ficheros, ya que no se puede acceder a ellos mientras que el `/home` del servidor esté montado. Si los clientes no pueden acceder al servidor, aparecerán de nuevo los directorios locales.

- El montaje se puede realizar en interactivo mediante el comando

```
mount -t nfs -o timeo=20,intr servidor.us.es:/home /home
```

También podemos incluir la siguiente línea en el fichero `/etc/fstab`, asegurando así que el directorio `/home` del servidor se monte cada vez que se bote el cliente:

```
servidor.us.es:/home /home nfs timeo=20,intr
```

¡Después de un "shutdown" de todas las máquinas hay que arrancar el servidor primero y posteriormente los clientes ! ¿ Qué haremos con los directorios locales que ya no se ven ? Podemos dejarlos "escondidos" un tiempo, ya que así será posible recuperar ficheros de estos directorios. Para acceder a ellos hay que desmontar el `/home` del servidor:

```
umount /home
```

Nota: Un sistema de ficheros sólo puede ser desmontado si no está en uso. Después de desmontar el `/home` del servidor podemos acceder otra vez a los ficheros locales. Si un usuario realiza un login y el servidor no está disponible, el proceso de login "aterrizará" en el `/home/usuario` local.

- Por esta razón, es conveniente avisar al usuario de este hecho, por ejemplo, mediante un fichero `.profile` en el `/home/usuario` de la forma:

```
echo ------------------------------------------------------
echo Al parecer el servidor no esta disponible.
echo Se encuentra en el directorio /tmp de la maquina local.
echo Por favor, transfiera su trabajo al servidor en cuanto
echo este disponible
echo ------------------------------------------------------
cd /tmp
```

Hemos mandado al usuario al directorio `/tmp`, porque así será fácil copiar algunos ficheros locales desde este directorio al servidor, en cuanto éste esté disponible (Los directorios locales en `/home` entonces ya no se verán !).

Notas:

Información sobre la carga de la red

Si hemos montado el servidor de ficheros, podremos conectarnos con la máquina menos cargada de la red, cada vez que hagamos un login. Siempre vamos a "aterrizar" en el mismo directorio de trabajo y, por lo tanto, siempre "veremos" los mismos ficheros. Esto permitirá un uso muy eficaz de todas las máquinas. ¿ Como podemos saber cuál es la máquina menos cargada en ese momento ? Hay varios métodos para conseguir esto, aquí proponemos una posible solución:

- En cada máquina de la red (clientes y servidor) activamos el demonio `rwhod`. Este programa recoge información sobre todas las máquinas de la red local que también tengan activadas este demonio. Podemos incluir las líneas

```
if [ -f ${NET}/rwhod ]
     then
     echo -n " rwhod"
     ${NET}/rwhod -t -s
fi
```

en el fichero `/etc/rc.d/rc.inet2`. Al botar la máquina, el demonio `rwhod` se activará y esta máquina dará información sobre su estado a otras máquinas de la red local. Con el comando

```
rwho
```

obtendremos información sobre todos los usuarios que estén trabajando en ese momento. El comando

```
ruptime
```

da información sobre todas las máquinas de la red local, el número de usuarios conectados con cada máquina, el tiempo que lleva encendida cada máquina desde el último arranque y la carga de cada máquina hace 1, 5 y 15 minutos. Aparecerán todas las máquinas de la red local que tengan activado el demonio `rwhod` . Para la comodidad del usuario podemos preparar un comando que extraiga sólo la información sobre las máquinas de nuestra red:

- Incluimos un alias en el `.profile` de cada usuario de la forma siguiente

```
alias redinfo='ruptime | grep -E
(cliente1|cliente2|servidor)'
```

El comando `redinfo` dará la misma información que `ruptime`, pero sólo para las máquinas `cliente1`, `cliente2` y `servidor`. Al realizar un login, cada usuario se puede informar sobre el estado de la red y conectarse con la máquina adecuada.

Notas:

6 Un servidor de la WWW

Hoy en día, muchas compañías comerciales e instituciones públicas mantienen una página en la WWW (World Wide Web). Las páginas de la WWW representan una posibilidad fácil e intuitiva de acceder a la información de la Internet. En esta sección vamos a aprender cómo "funciona" el WWW, y como podemos publicar una página en la WWW usando un PC con Linux que está conectado a la red.

Cómo funciona la WWW

La Internet básicamente es una red de ordenadores mundial, todos los ordenadores que forman parte en ella pueden comunicarse entre sí. Hay varias formas de intercambiar información (`mail, news, ftp, telnet, etc.`), una de ellas es la WWW. El concepto de la WWW es el siguiente

- Cada persona/empresa que quiere publicar información, prepara esta información en un cierto formato (HTML = Hypertext Markup Language).
- Esta información se guarda en un ordenador que está conectado a la red. Un programa que se encuentra siempre en la memoria del ordenador (el servidor `httpd`) permite el acceso de otros miembros de la red a esta información. El conjunto de todos estos ordenadores se llama "servidores de la WWW".
- Usuarios en todo el mundo pueden conectarse con la red y, usando un "browser" (como Netscape o Mosaic), pueden acceder a toda esta información. En el browser, el usuario indica el nombre del ordenador y el nombre del documento al que quiere acceder. El ordenador se "pone en contacto" con el servidor de la WWW, y el servidor suministra la información (el documento) deseada.
- Las páginas de la WWW pueden incluir gráficos, sonidos, vídeos, aplicaciones "java" etc. El texto (Hypertext) permite usar "links", un simple click con el ratón sobre un link nos lleva a otro documento. Este documento puede estar situado incluso en otro ordenador.

Aquí vamos a explicar cómo podemos publicar una página de la WWW en nuestro PC con Linux. Esto require la instalación del servidor `httpd`. Una vez instalado y configurado el servidor, escribiremos nuestra pequeña página de la WWW.

Notas:

Usar el httpd del Slackware 3.0

En la distribución del Slackware 3.0, podemos instalar el servidor de la WWW a la hora de instalar el Linux. De esta manera podemos prescindir de traer este programa por la red. Sin embargo, la configuración del servidor de la WWW, tal y como viene instalado en el Slackware 3.0 no es correcta. Al arrancar el ordenador podemos ver el mensaje de error:

```
Syntax error on line 37 of /var/lib/httpd/conf/httpd.conf:
ServerRoot must be a directory
```

Si analizamos los ficheros de configuración, nos daremos cuenta de que el servidor `httpd` está configurado para residir en el directorio `/usr/lib/httpd`. No obstante, el programa está instalado en el directorio `/var/lib/httpd`. Editaremos todos los ficheros `*.conf` en el directorio `/var/lib/httpd/conf`, y cambiaremos todas las apariciones de "`/usr`" por "`/var`".

Ahora arrancaremos el ordenador de nuevo e intentaremos conectarnos con un "browser" a la dirección "httpd://miordenador.donde.esta/", en nuestro caso "httpd://idefix3.us.es". Si todo está bien, saldrá una página de la WWW muy simple.

Advertencia: Siempre y cuando el servidor de la WWW esté activo en nuestro sistema, personas de todo el mundo podrán acceder a nuestra máquina. Debemos comprobar la configuración de este programa para controlar el accesso a nuestro sistema. De hecho, en el servidor `httpd` que acabamos de activar, hay un "agujero de seguridad" que debemos "cerrar".

Notas:

Instalación del servidor `httpd`

Si no está disponible el servidor de la WWW en nuestra distribución de Linux, tendremos que traernoslo por la red. Hay varios servidores `httpd` (d=daemon, demonio) diponibles en la red (CERN httpd, NCSA httpd, Apache httpd), aquí vamos a instalar el Apache `httpd`. Aquí en España podemos encontralo bajo la dirección "http://www.develnet.es/apache/". Si está disponible una versión binaria del programa para Linux, cogeremos el fichero correspondiente para nuestro sistema (Linux/ELF o Linux/a.out).

Al descomprimir el fichero `apache_1.1-linux-ELF.tar.gz`, crearemos un subdirectorio, llamado `apache_1.1` que contendrá todos los ficheros necesarios. Abriremos el fichero `README` con un editor. Este fichero contiene una descripción del programa e indica los pasos que hay que seguir para su instalación.

Configuración del servidor de la WWW

Una vez instalados los ficheros tenemos que configurar el programa. En el caso del Apache `httpd` hay tres ficheros de configuración:

```
httpd.conf
access.conf
srm.conf
```

Vamos a editar estos ficheros y a discutir brevemente los parámetros más importantes. Para una descripción detallada, véase "http://www.develnet.es/apache/docs/". Recordemos que al instalar el `httpd` abrimos una "puerta" de nuestro ordenador al mundo. ¡Es importante que entendamos los cambios que realizamos! Los ficheros de la distribución proporcionan una configuración bastante segura, pero hay que tener cuidado a la hora de ampliar esta configuración.

Notas:

El fichero `httpd.conf`

En este fichero configuraremos el funcionamiento general de nuestro servidor:

- Las variables `ServerType`,`Port`, `User` y `Group` no requieren cambios.
- `ServerAdmin`. Aquí pondremos la dirección de e-mail del responsable del servidor.
- `ServerRoot`. Es el directorio de donde "cuelgan" los directorios `conf` y `logs`. Aquí usaremos el mismo directorio que en la distribución del Slackware 3.0, esto es, el directorio `/var/lib/httpd`.
- Las variables `ErrorLog` y `TransferLog` determinan los nombres de los ficheros "log". Estos dos ficheros serán muy útiles a la hora de comprobar quién ha entrado en nuestro servidor y qué errores han surgido (¡muy útil para el desarrollo de páginas!)
- En una máquina pequeña podemos limitar el número máximo de clientes mediante la variable `MaxClients` (normalmente 150).

Notas:

El fichero `access.conf`

Ahora determinaremos los directorios a los cuales se puede acceder:

- El directorio "raiz" de los documentos podría tener una entrada como la siguiente:

```
<Directory /usr/local/etc/httpd/htdocs>
# This may also be "None", "All", or any combination of "Indexes",
# "Includes", "FollowSymLinks", "ExecCGI", or "MultiViews".
Options FollowSymLinks ExecCGI
# This controls which options the .htaccess files in
# directories can override
AllowOverride None
# Controls who can get stuff from this server.
<Limit GET>
order allow,deny
allow from all
</Limit>
</Directory>
```

  Esta configuración permite la ejecución de un "cgi-script". Hay que tener cuidado con la instalación de estos scripts (véase siguiente sección). Si usamos la versión 1.1.1 del Apache `httpd` debemos quitar la opción "Indexes", ya que esta opción puede proporcionar accesso a personas no autorizadas.

- Podemos configurar un directorio de accesso restringido de la siguiente manera:

```
<Directory /var/lib/httpd/htdocs/limited>
AllowOverride None
# Controls who can get stuff from this server.
<Limit GET>
order deny,allow
deny from all
allow from servix.us.es
</Limit>
</Directory>
```

Notas:

El fichero `srm.conf`

En este fichero configuraremos

- `DocumentRoot`. Este es el directorio que será visible desde "fuera". Aquí usaremos el directorio `/var/lib/httpd/htdocs`.
- `DirectoryIndex`. Este es el nombre del fichero que debe residir en el directorio `DocumentRoot` y en todos sus subdirectorios. Aquí usaremos el nombre `index.html`.

Depués de configurar el servidor, mataremos todos los procesos `httpd`, por ejemplo, mediante:

```
ps -uax | grep httpd
kill ..
kill ..
```

empezando por el proceso con el número más bajo. Ahora podremos arrancar de nuevo el servidor mediante el comando (depende de la instalación):

```
/usr/sbin/httpd&
```

Probaremos si todas las configuraciones funcionan como queríamos.

Notas:

Nuestra pequeña página de la WWW

Una vez que tengamos el httpd configurado y activo, podemos publicar nuestra primera página. Las páginas de la WWW se escriben en un formato llamado HTML (Hypertext Markup Language). Aquí no podemos explicar el lenguaje HTML. En su lugar, ofreceremos el código fuente de la página que presentamos en este curso.

Abriremos un fichero con el nombre `index.html` e incluiremos, por ejemplo:

```
<html>
<head>
<title>Bienvenidos al servidor del curso !</title>
</head>
<h1>Bienvenidos al servidor del curso!</h1>
<body>
Hola, este es el servidor del curso de Linux. Para que se vea algo,
aquí hay un gráfico:
<p><img src="./bfanim.gif" width=300><p>
<a href="http://numerix.us.es/curso.html">Material</a> del curso
de Linux.
<p>
<a href="http://numerix.us.es/group">Documentación</a> del sistema, solo
para miembros del grupo de investigación.
<p>
<hr>
Aqui se suele poner la dirección de email del responsable de
la pagina, por ejemplo:
<p>
Comentarios a <a href="mailto:denk@numerix.us.es">Claus Denk</a><br>
</body>
</html>
```

Ahora copiaremos este fichero al directorio `/var/lib/httpd/htdocs`. Finalmente, es conveniente comprobar si todo funciona: Abrimos un "browser" y entramos en la dirección de nuestro ordenador "http://idefix3.us.es/". Ya debe aparecer nuestra página.

Los scripts CGI

En el directorio `/var/lib/httpd/cgi-bin` encontraremos algunos shell-scripts que pueden ser ejecutados desde la página WWW de nuestro servidor. Podemos, por ejemplo, acceder a la dirección "http://idefix3.us.es/cgi-bin/uptime" para ejecutar uno de estos scripts. ¡Ojo, estos scripts se ejecutan en nuestro ordenador y, por tanto, debemos comprobar la seguridad de todos los scripts que instalamos!

Direcciones útiles

- "http://www.w3.org/pub/WWW/MarkUp/MarkUp.html". Estandár HTML y varios "links" para aprender HTML.
- "http://www.smeal.psu.edu/misweb/language/htmlmain.html". Introducción a HTML.
- "http://www.vol.it/MIRROR/EN/images/". Colección de imagenes.
- "http://homepage.seas.upenn.edu/ mengwong/perlhtml.html". Perl scripts y HTML.

7 Ordenadores de alto rendimiento con Linux

En esta sección daremos una introducción general al tema "high performance computing". Vamos a ver cómo podemos "construir" un ordenador de alto rendimiento usando varias estaciones de trabajo. Dirigiremos nuestro interés al cálculo numérico, tal y como aparece en la Física o en la Ingeniería. La velocidad de una CPU (en combinación con su memoria y el "bus") depende fundamentalmente de las siguientes características:

- Frecuencia de la CPU.
- Cuántas instrucciones pueden realizarse a la vez.
- Velocidad de acceso a la memoria y a la memoria "cache"

Todos estos parámetros están sometidos a límites físicos. Para construir ordenadores de alto rendimiento se aplican los siguientes "trucos" encaminados a sortear estas dificultades:

- Arquitecturas vectoriales. Estos ordenadores trabajan a frecuencias de CPU bastante bajas, pero son capaces de realizar un gran número de operaciones vectoriales a la vez.
- Arquitecturas paralelas "shared memory". Varios CPU's trabajan en paralelo, accediendo a una memoria común.
- Arquitecturas paralelas "distributed memory". Varios CPU's, cada una con su propia memoria se "reparten" el trabajo

Notas:

Podemos usar el Linux para las dos últimas configuraciones:

- Linux SMP ("Symmetric Multi Processor"). Existen placas que soportan hasta 4 CPU's Pentium o Pentium Pro. Los procesadores pueden acceder a la misma memoria.
- Un "cluster" de ordenadores con Linux. Varios ordenadores independientes se comunican entre ellos para poder realizar un trabajo en común.

Aquí vamos a tratar el último caso, ya que es el más sencillo.

¿Cómo funciona un cluster?

En varios ordenadores siempre podemos ejecutar varios programas a la vez. La dificultad surge si queremos ejecutar UN programa en varios ordenadores a la vez. Podemos ver con un ejemplo cómo se puede realizar esta tarea: si queremos sumar, digamos $N = 1000$ elementos de un vector en un cluster de $n_{nodes} = 4$ ordenadores, se podría crear un programa que realizara lo siguiente:

- El programa se arranca en un ordenador, la memoria de este ordenador contiene los 1000 elementos del vector (proceso "0")
- Ahora se arrancan 3 procesos más en los otros ordenadores, cada de los cuales se queda esperando una señal del proceso 0.
- El proceso 0 "manda" los elementos del vector con índices 251 .. 500 al proceso 1, elementos con índices 501 .. 750 al proceso 2 y elementos con índices 751 .. 1000 al proceso 3
- El proceso 0 empieza a sumar "sus" elementos (1..250). Los otros procesos empiezan a sumar sus elementos independientemente.
- En el momento en el que un proceso ha realizado su suma, se lo comunica al proceso 0 y le manda el resultado de la suma.
- El proceso 0 se encarga de sumar su resultado con los resultados de los demás procesos para hallar la suma total.

Notas:

En la práctica habría que comprobar si nuestro programa es de verdad más rápido en cuatro ordenadores que en uno. Esto dependerá del tipo de programa que estemos escribiendo y del hardware del que dispongamos. Este ejemplo nos lleva a introducir varios conceptos importantes:

- Paralelismo inherente al algoritmo. Existen algoritmos "course grained" y "fine grained". Estos dos tipos de algoritmos se diferencian en la cantidad de código que puede realizar de manera independiente cada máquina.
- "Scaling" del algoritmo en el cluster. Se mide cómo depende el tiempo de ejecución del número de "nodos" que usamos en el cluster. En nuestro ejemplo, la parte más pequeña que se puede realizar en un procesador es la suma de dos elementos del vector. Por lo tanto, el número máximo de nodos de nuestro cluster sería 500. El trabajo que tiene que realizar cada nodo es una suma de $N/n_{nodes} - 1$ sumas. Necesitaríamos un sistema de comunicación muy eficaz para poder usar varios nodos con el máximo rendimiento.

El sistema de comunicación viene caracterizado por dos parámetros:

- "bandwidth". La cantidad de datos que se puede transferir por unidad de tiempo una vez que la transmisión ha empezado. Se mide en Mbits/s.
- "latency". El tiempo mínimo necesario para transferir un objeto a otro nodo. Se mide en μs. Este parámetro es el más importante.

Aquí citamos algunas posibilidades de comunicar las máquinas de un cluster:

Nombre	Tipo	bandwidth (Mbit/s)	latency (μs)	precio/nodo (ptas)
ParaStation	Tarjeta PCI	125.0	2	220000
Ethernet	Tarjeta PCI/ISA	10.0	1000	5000
TTL_PAPERS	Puerto paralelo	1.6	3	8000

En este curso vamos a usar la conexión vía Ethernet como medio de comunicación. Para un cluster más avanzado sería aconsejable un medio de comunicación con una "latency" más baja.

Notas:

Ventajas e inconvenientes de un cluster

Un cluster de ordenadores tiene varias ventajas frente a otros sistemas de computación paralela:

- Cada máquina puede ser utilizada independientemente como estación de trabajo.
- Los componentes necesarios del hardware son baratos.
- Es posible un grán número de nodos, de forma que el ordenador puede "crecer" con las necesidades. ¡ Existen clusters con 12.000 nodos !
- El mantenimiento es sencillo.
- Ya existe software preparado para clusters.

Sin embargo, no es trivial la programación de un código preparado para la ejecución paralela.

Se puede obtener más información acerca de la computación paralela con Linux en la dirección "http://yara.ecn.purdue.edu/ pplinux/".

Intercambio de mensajes - MPI

Para la comunicación entre los nodos es necesario un sistema de mensajería. Hay dos sistemas disponibles:

- PVM (Parallel Virtual Machine). Es el estándar "de facto".
- MPI (Message Passing Interface). Es el estándar oficial.

Ambos proporcionan un "interface" para el programador que permite intercambiar mensajes y realizar operaciones básicas entre los nodos del cluster. Hay una versión de libre distribución del MPI, llamada MPICH. Podemos obtenerla en la dirección "http://www.mcs.anl.gov/Projects/mpi/mpich/". Aquí vamos a suponer que tenemos

- Un servidor de ficheros montado que permita usar el mismo sistema de ficheros en todos los nodos del cluster.
- El MPICH instalado.

Un ejemplo para la programación con el MPI

Ahora vamos a ver cómo se paraleliza un programa simple. El programa original es:

```
        PROGRAM EULER
C ======================================================================
C       Serial version of Euler´s number.
C ======================================================================
        DOUBLE PRECISION GAM
        GAM = 1.0d0
        WRITE(*,*) ´Input NMAX´
        READ(*,*) NMAX
        WRITE(*,*) ´doing ´,NMAX,´ iterations ..´
        DO I=2,NMAX
          GAM=GAM+1.0d0/dble(I)
        ENDDO
        GAM = GAM - DLOG(dble(NMAX))
        WRITE(*,10) GAM
        WRITE(*,*) ´exact :0.57721566490153286060´
 10     FORMAT(1x,´result:´,E20.14)
        END
```

Este programa realiza lo siguiente:

- Inicializar la variable GAM a 1.0.
- Leer la variable NMAX del "standard input".
- Calcular la suma $\gamma = \sum_{i=1}^{N_{max}} 1/i - ln(N_{max})$.
- Imprimir el resultado en el "standard output".

Notas:

Para paralelizar este código tenemos que considerar:

- Que la salida y la entrada de datos sólo se realiza en un nodo.
- Que podemos realizar la suma en n_{nodes} trozos
- Que sólo hay que comunicar la variable NMAX y los resultados de las sumas entre los nodos.

El resultado, usando el MPI es:

```
        PROGRAM TESTMPI
C ================================================================
C       parallel version of euler´s number using MPI
C ================================================================
        include ´mpif.h´
        DOUBLE PRECISION GAM,GAMSUM,stime,etime
C
C       Init MPI
C
        call MPI_INIT( ierr )
        call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
        call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
        print *, "Process ", myid," is running ..."
C
C       Only process 0 prints, reads NMAX and starts timer
C
        IF (myid.eq.0) THEN
          WRITE(*,*) ´Input NMAX´
          READ(*,*) NMAX
          WRITE(*,*) ´doing ´,NMAX,´ iterations ..´
          stime = MPI_WTIME()
        ENDIF
C
C       broadcast NMAX to all processes
C
        call MPI_BCAST(NMAX,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
        if ( NMAX .le. 0 ) goto 30
C
C       each process does his work
C
        GAM=0.0d0
```

```
        DO I=2+myid,NMAX,numprocs
          GAM=GAM+1.0d0/dble(I)
        ENDDO
C
C       sum GAM of each process into GAMSUM
C
        call MPI_REDUCE(GAM,GAMSUM,1,MPI_DOUBLE_PRECISION,MPI_SUM,0,
     $     MPI_COMM_WORLD,ierr)
C
C       only process 0 writes output and calculates wall time
C
        IF (myid.eq.0) THEN
          GAMSUM = GAMSUM + 1.0d0 - DLOG(dble(NMAX))
          WRITE(*,22) GAMSUM
 22       FORMAT(1x,´result:´,E20.14)
          WRITE(*,*) ´exact: 0.57721566490153286060´
          etime = MPI_WTIME()
          WRITE(*,*) ´wall clock:´,etime-stime
        ENDIF
C
C       Exit point
C
 30     call MPI_FINALIZE(rc)
        END
```

Notas:

A la hora de ejecutar el programa con el MPI, el MPI crea una copia del código y lo ejecuta en cada nodo del cluster. Vamos a analizar este código paso a paso. Inicializaremos el MPI:

```
call MPI_INIT( ierr )
call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
print *, "Process ", myid," is running ..."
```

La llamada a MPI_COMM_RANK devuelve un número de identificación (myid) para cada proceso. El número de procesadores (numprocs) se determina mediante una llamada a MPI_COMM_SIZE. Cada proceso imprime un mensaje cuando se ha inicializado. Ahora, el proceso 0 lee la variable NMAX del "standard input". El proceso 0 es un proceso especial, denominado proceso "root". Este proceso se ejecuta en la máquina donde hemos arrancado el programa. Con el comando

```
call MPI_BCAST(NMAX,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
```

el proceso 0 "comunica" el valor de NMAX a todos los demás procesos. A partir de este punto, cada proceso realiza "su" parte de la suma. ¿ Por qué hemos inicializado GAM a zero en este caso ? Con la llamada

```
   call MPI_REDUCE(GAM,GAMSUM,1,MPI_DOUBLE_PRECISION,MPI_SUM,0,
$     MPI_COMM_WORLD,ierr)
```

sumamos los resultados GAM en la variable GAMSUM del proceso 0. Ahora, el proceso 0 puede calcular el resultado e imprimirlo en el "standard output".

Notas:

Usar el MPI con el LINUX

A la hora de instalar el MPICH se han creado varios scripts que utilizaremos para compilar y ejecutar un programa. Mediante el comando

```
mpif77 -o ejecutable fichero.f
```

compilamos el fichero "fichero.f" (salida "ejecutable"). Para ejecutar el programa, usaremos

```
mpirun -np N ejecutable
```

Aquí N es el número de nodos que queremos usar en el cálculo. El número de nodos usados puede ser mayor que el número de nodos disponibles para poner a prueba el programa. En este caso MPICH ejecutará más de un proceso en cada nodo.

¿ Qué máquinas usará el MPICH para ejecutar el programa ? En el árbol de directorios del MPICH, existe un fichero llamado "machines.LINUX", donde podemos enumerar las máquinas de nuestro cluster. En nuestro cluster de la Facultad de Física, disponemos de las siguientes máquinas:

numerix	P133
idefix1	P133
frodo	P133
servix	P100

El tiempo de ejecución de nuestro ejemplo con NMAX=100000000 es:

versión serial, P133	40 s
versión paralela, 1 nodo P133	45 s
versión paralela, 2 nodos P133	22 s
versión paralela, 3 nodos P133	15 s
versión paralela, 3 nodos P133 + 1 nodo P100	14 s

Aquí vemos, que la lentitud del P100 "frena" el programa, ya que le hemos mandado el mismo trabajo que a los demás nodos. Un modelo de programación avanzado debe tener en cuenta la capacidad de cálculo de cada nodo para evitar este problema ("load balancing"). Debido al hecho de que cada nodo puede realizar mucho trabajo independientemente (15 s), el "scaling" de nuestro programa con el número de nodos es bastante bueno.

¡ Happy computing !